BLACK CODHER

CODING PROGRAMME

CODING BLACK FEMALES

Black Codher Bootcamp

NÍ YỌ

BLACK CODHER

CODING PROGRAMME

UNIT 4 - Session 5 React

End of Session 4 Summary

In the last session we covered the following:

1. Understanding React Hooks
   - useState
   - useEffect

2. Understanding JSON files

3. Understand Object and Array deconstruction

4. Creating your first JSX component

5. Identify the components in your Library React App

Goals for Unit 4 - Session 5

1. Reviewing Bookcase App Homework
2. Nested components
3. Using Proptypes for better validation and type checking
4. Lifting State Up
5. Handling Events
6. Continuing with the Bookcase App

Bookcase App Review

Add the Book Component

- In the **mybookcase** app we have created, each object in the **books.json** file can be translated logically into a component

- In the previous session/homework we added a **Book.js** file which contained a functional Book component:

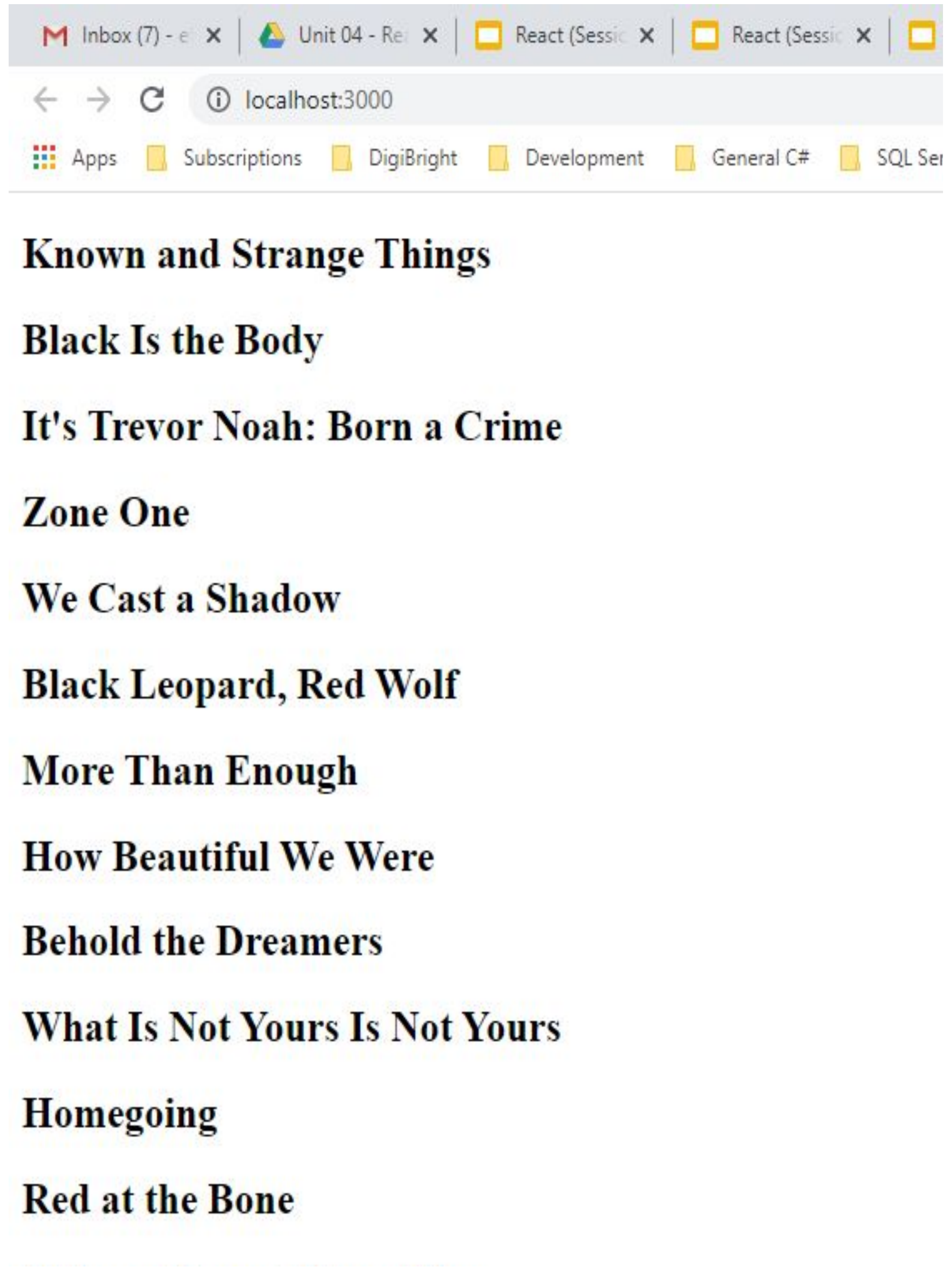
```
import React from 'react';

const Book = (props) => {
   return (
         <div>
           <h2>{props.book.volumeInfo.title}</h2>
         </div>
   );
}

 export default Book;
```

Book Component Rendered

- In the browser, on the Elements tab of the developer tools, the output of the list of <Book/> components translates to a list of **<div>** and **<h2>** tags



Known and Strange Things

Black Is the Body

It's Trevor Noah: Born a Crime

Zone One

We Cast a Shadow

Black Leopard, Red Wolf

More Than Enough

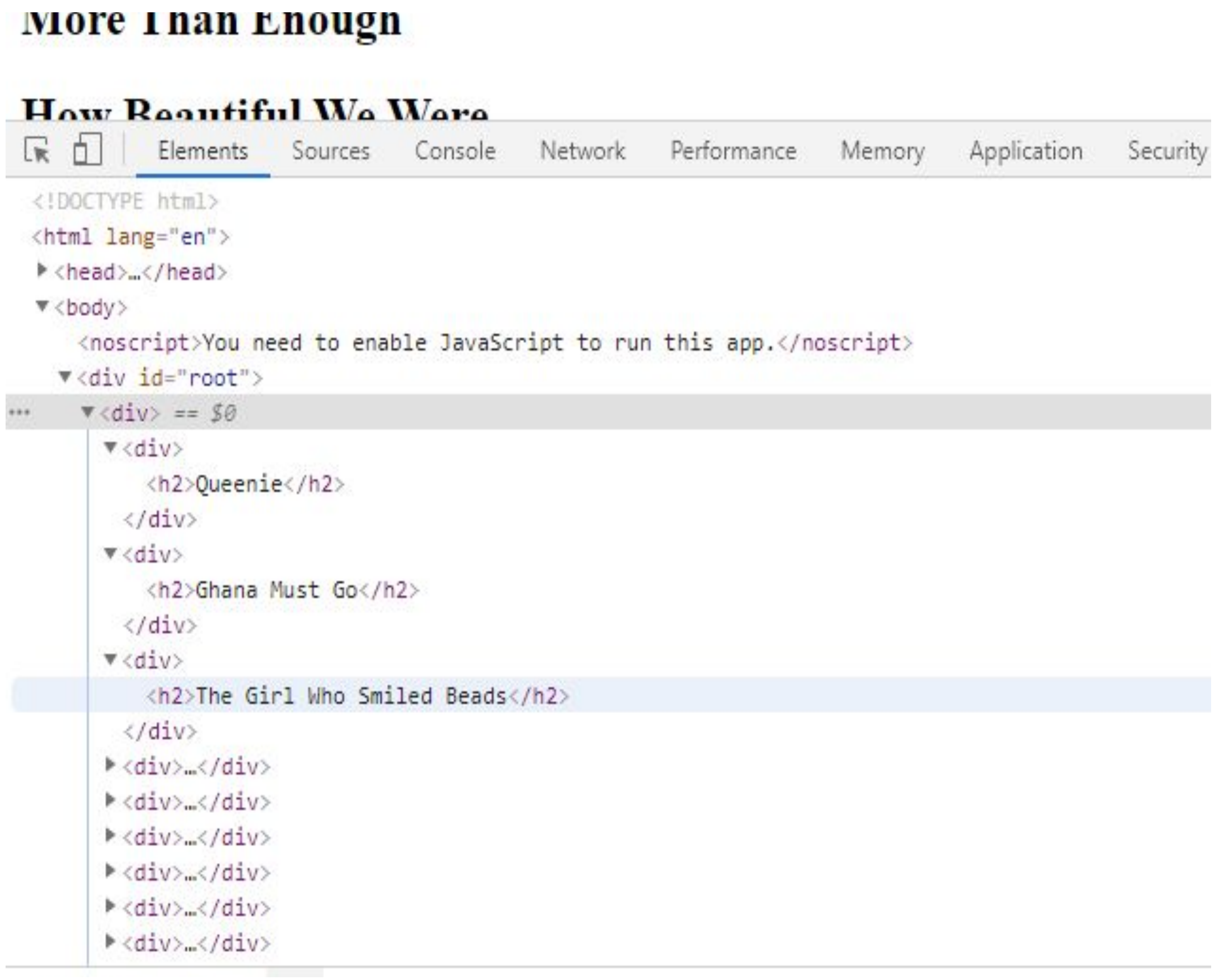
How Beautiful We Were

Behold the Dreamers

What Is Not Yours Is Not Yours

Homegoing

Red at the Bone



```
<!DOCTYPE html>
<html lang="en">
 ▶<head>…</head>
 ▼<body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
   ▼<div id="root">
...  ▼<div> == $0
      ▼<div>
         <h2>Queenie</h2>
       </div>
      ▼<div>
         <h2>Ghana Must Go</h2>
       </div>
      ▼<div>
         <h2>The Girl Who Smiled Beads</h2>
       </div>
      ▶<div>…</div>
      ▶<div>…</div>
      ▶<div>…</div>
      ▶<div>…</div>
      ▶<div>…</div>
      ▶<div>…</div>
```

Book Component Props

- The title of the book uses the **props.book.volumeInfo.title** attribute(line 6 **Book.js** file)

```
5 ...
6<h2>{props.book.volumeInfo.title}</h2>
7 ...
```

- React uses components to divide the UI (User Interface) into little reusable pieces of code. In some cases, those components need to send data to each other. **The way to pass data between components is by using props**.

- The **props** of the Book component is passed down by the calling/parent component, which is declared in the **App.js** file

App Component

- The App component declared in **App.js** passes each individual book object from the **books.json** to the child via the child's book attribute (highlighted)

```
import React, { useState } from 'react';
import Book from './components/Book';
import data from './models/books.json';

const App = (props) => {
const [books] = useState(data);

return (
     <div>
         {books.map(book => <Book key={book.id} book={book}/>)}
     </div>
 );
}
export default App;
```

Book Props Deconstructing

- Back to the Book component in Book.js. Deconstructing the props.book to display the **title**, **price**, **authors** and **images** could look like this:

```
const {id, saleInfo:{retailPrice}, volumeInfo: {title, authors,
description, imageLinks: {thumbnail}}} = book;
```

- This will mean you can reference each property using their variable names:

```
<div className="book">
  <img src={thumbnail}/>
   <div>
     <h2 title={title}>{title}</h2>
   </div>
</div>
```

Conditional Rendering

Conditional Rendering

- It's important to note that we **can't use if/else statements inside a functional JSX component declaration**.

- **Conditional logic** or **ternary operators** have to be used to short-circuit evaluation

Ternary Operator

- In our Book.js some of the properties of the book object could be empty. This would cause an error when running the app. In order to display them we will need to use **ternary** operators to test if the variable is present before displaying:

```
Syntax: <boolean condition> ? <truthy expression> : <falsy expression>
e.g. price ? `£{price}` : 'No price listed'
```

Conditional Rendering

- An example of using ternary logic in our Book.js file:

```
<p>by {authors ? authors.join(', '):"No Authors Listed"}</p>
```

- If the authors variable is not empty, display a comma separated list of authors else display the string "No Authors Listed"

- Another example of conditional rendering

```
return (
    <div className="list">
      <div>
       {(props.stored==="library") && <h2>Suggested Reading</h2>}
      </div>
    </div>
  );
```

Conditional Rendering

- A short circuit example in a **BookList** component.

```
const BookList = ({books,...props}) => {

  if(books.length === 0){
    return <div className="empty">No books...</div>;
  }

  //Standard return logic
  return( … );
}
```

- If there are no books in the books array then display the text "No Books". If there are books in the array the next return statement will be run

BLACK CODHER

CODING PROGRAMME

Prop Types

PropTypes

- As the App grows, they may be an increasing amount of bugs. Some of these may be down to type checking errors. To help with checking types you can import the **prop-types** library

```
import PropTypes from 'prop-types';
```

- It contains validators that can help ensure your data is of the correct type/shape. Some of the validators are listed below

```
.IsRequired
.array
.number
.string
.shape
```

- It's possible to chain and nest the validators in one command e.g. `array.IsRequired`

PropTypes

- Using the **book.json** data as an example we know each book should have a **title**, an array of **authors**, a **description** and a **price**.

- Using an example book object with the following shape, the prop types could be set before the final export in the following way:

Example Book Object:

```
{
    "volumeInfo": { "title": "Queenie" },
    "authors": ["Candice Carty-Williams"],
    "description": "",
    "price":"7.99",
}
```

PropTypes

Example Book Prop Types:

```
Book.propTypes = {
  volumeInfo: ProtoTypes.shape({title: PropTypes.string.isRequired}),
  authors: PropTypes.array.isRequired,
  description: PropTypes.string,
  price: PropTypes.number.isRequired
};

Book.defaultProps = { description: "No description…"};
```

- PropTypes exports a range of validators that can be used to make sure the data you receive is valid.

Default PropTypes

- When an invalid value is provided for a prop, a warning will be shown in the JavaScript console.

- It's important to note that the application will still run but the warning will also be displayed in the browser console

- It is also possible to set a default property, this is useful for when the data returned is null or empty for some object

```
Book.defaultProps = {
    price: 'No price provided'
};
```

Exercise 1

Exercise 1: Adding Type Checking

- In the **mybookcase** app add type checking for the following **Book** components (Book.js)

1. title
2. authors
3. price
4. description
5. thumbnailImage

- Add default attributes using the defaultProps

- Open the browser or view the console app to see the warning messages returned by the vaildators

Checkpoint!

How are you feeling?

RED - I have no idea what you're talking about.

YELLOW - I have some questions but feel like I understand some things.

GREEN - I feel comfortable with everything you've said.

BLACK CODHER

CODING PROGRAMME

Nested Components

Nested Components

- A lot of the power of React is its ability to allow nesting of components

- Nesting components makes it easy and fast to separate UI elements and easy to inject **props** down to children based on the parent component's state

- There are a few ways to nest components:

1. Nesting using props.children (Containment)
2. Nesting with child components (Specialization)

Nesting with props.children

- In the example below the special prop "**children**" is used to nest the child components. props.children

```
function BookList(props) {
  return (
    <div className={booklist booklist-' + props.color}>
      {props.children}
    </div>
  );
}
```

- The BookList doesn't need to know or manage the state of the components in the **props.children**

Nesting with props.children

- The **props.children** lets other components pass arbitrary children to a component by nesting the JSX, e.g.

```
function App() {
   const [books] = useState(data);
  return (
    <BookList color="orange">
      <div>
        <h2>Synopsis Viewer</h2>
         <p>Click on the book to see the synopsis here...</p>
      </div>
      {books.map(book => <Book key={book.id} book={book}/>)}
     </BookList>
  );
}
```

Nesting with props.children

- The lines highlighted below will be rendered in BookList when **props.children** is called

```
function App() {
  return (
    <BookList color="green">
      <h2 className="book-title">{book.title}</h2>
      <p className="book-description">{book.description}</p>
    </BookList>
  );
}
```

Nesting with Specialization

- Instead of inheritance we can use nesting with specialization to create special versions of components reusing the same components

```
function Message(props) {
  return (
    <Container color="blue">
      <h1 className="message-title">
        {props.title}
      </h1>
      <p className="message-text">
        {props.text}
      </p>
    </Container>
  );
}
```

```
function WelcomeMessage() {
  return (
    <Message
      title="Welcome"
      text="Thank you for visiting our
spacecraft!" />
  );
}
```

Exercise 2

Exercise 2: Nesting your Books

- We are going to create a container for our bookcase called BookList. It will be responsible for rendering the books that are contained in **books.js**.
- In your **mybookcase** app, create a file in the **components** folder called **BookList.js**
- Add a functional component method called **BookList**.
- Update the **BookList.js** to return a <Book/> component. Ensure you pass a book attribute to the Book component
- Update your **App.js** file to reference the **BookList.js** and return the **<BookList/>** object. Ensure you are passing the collection of book objects to the BookList to make it available in the props

BLACK
CODHER

Handling Events

Handling Events

- Handling events with React is very similar to handling events on DOM elements with a few syntax changes.

- Events on React elements use camel**C**asing rather than lower case (JavaScript) e.g. onChange instead of onchange.

Plain HTML/Javascript:
```
<input type="text" name="search" id="search"
onchange="previewSearch()"/>
```

React:
```
<input type="text" name="search" id="search"
onChange={previewSearch}/>
```

Handling Events

- Another difference is you cannot return false to prevent default behaviour you must call **preventDefault** explicitly.

Plain HTML: `<a href="#" onclick="console.log('Hello');return false;">Say Hello</a>`

React:

```
function MyLink() {
   function handleClick(event){
      event.preventDefault();
     console.log('Hello');
   }
   return(
      <a href="#" onclick="{handleClick}">Say Hello</a>
   );
}
```

Exercise 3

Exercise 3: Event Handling

1. Open the **mybookcase** app

2. We should have a **App.js** file which returns a **BookList** component e.g.

```
const App = (props) => {
  const [books] = useState(data);
  return  <BookList books={books} />
}
```

3. Open the **Book.js** file edit the Book component. Add a button called "Add +"

4. Add a function to the button called `addBook()` pass the **title** as an argument:

```
onclick={() => addBook(title)}
```

Exercise 3: Event Handling

5. In the **Book.js** file add the following function:

```
function addBook(title) {
    console.log(`The Book ${title} was clicked`);
 }
```

6. Run the app. In the browser view the console window for the results of clicking the buttons. The title of the book should be logged in the console.

Checkpoint!

How are you feeling?

RED - I have no idea what you're talking about.

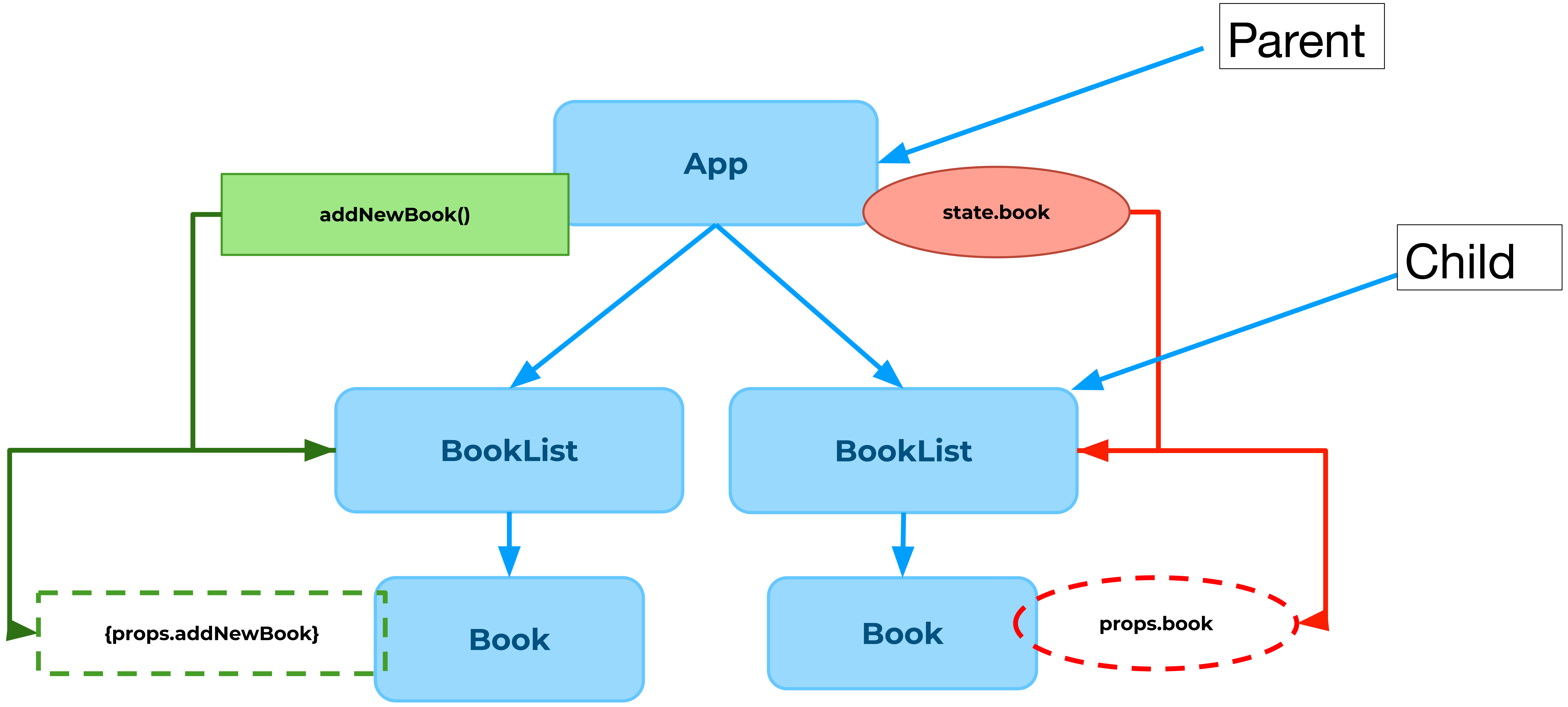
YELLOW - I have some questions but feel like I understand some things.

GREEN - I feel comfortable with everything you've said.

Lifting State Up

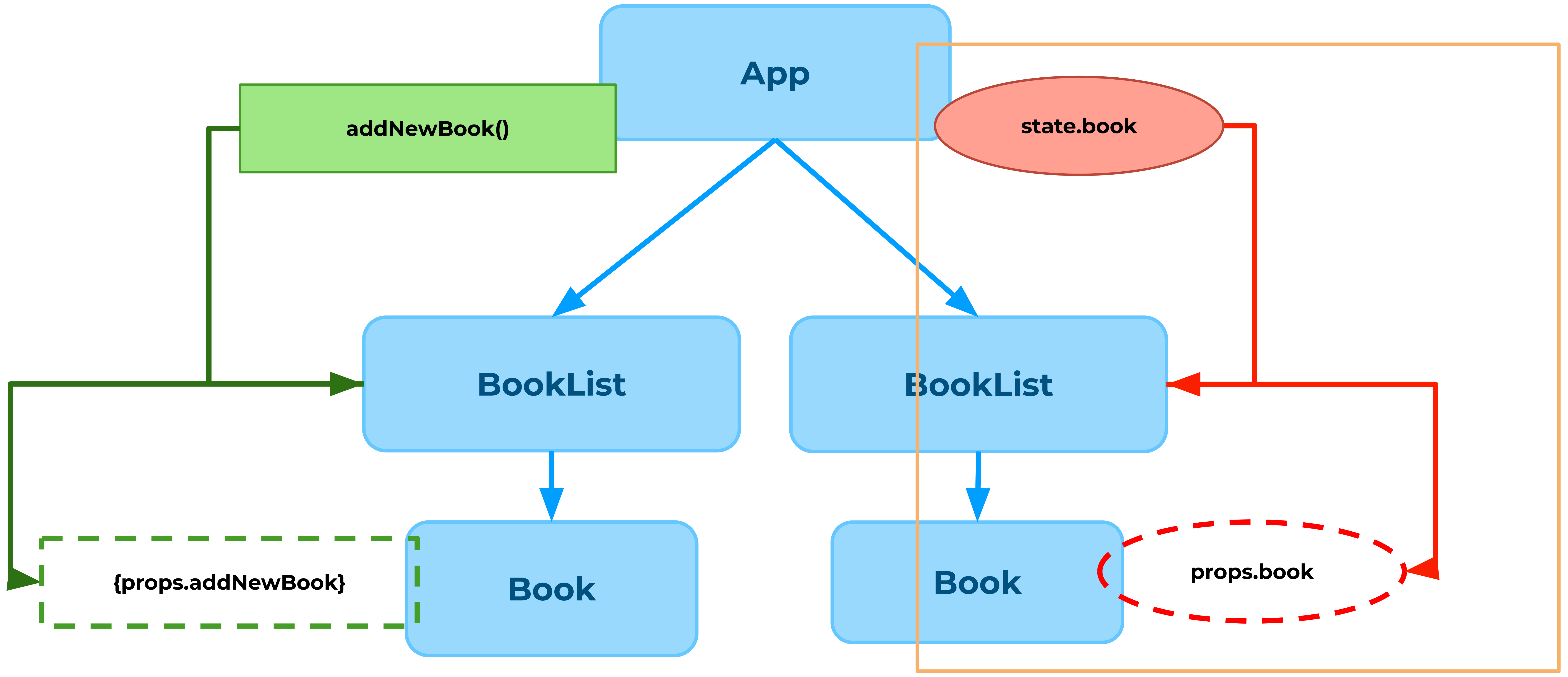
Lifting State Up

- In React, data can be passed from parent to child and from child to parent



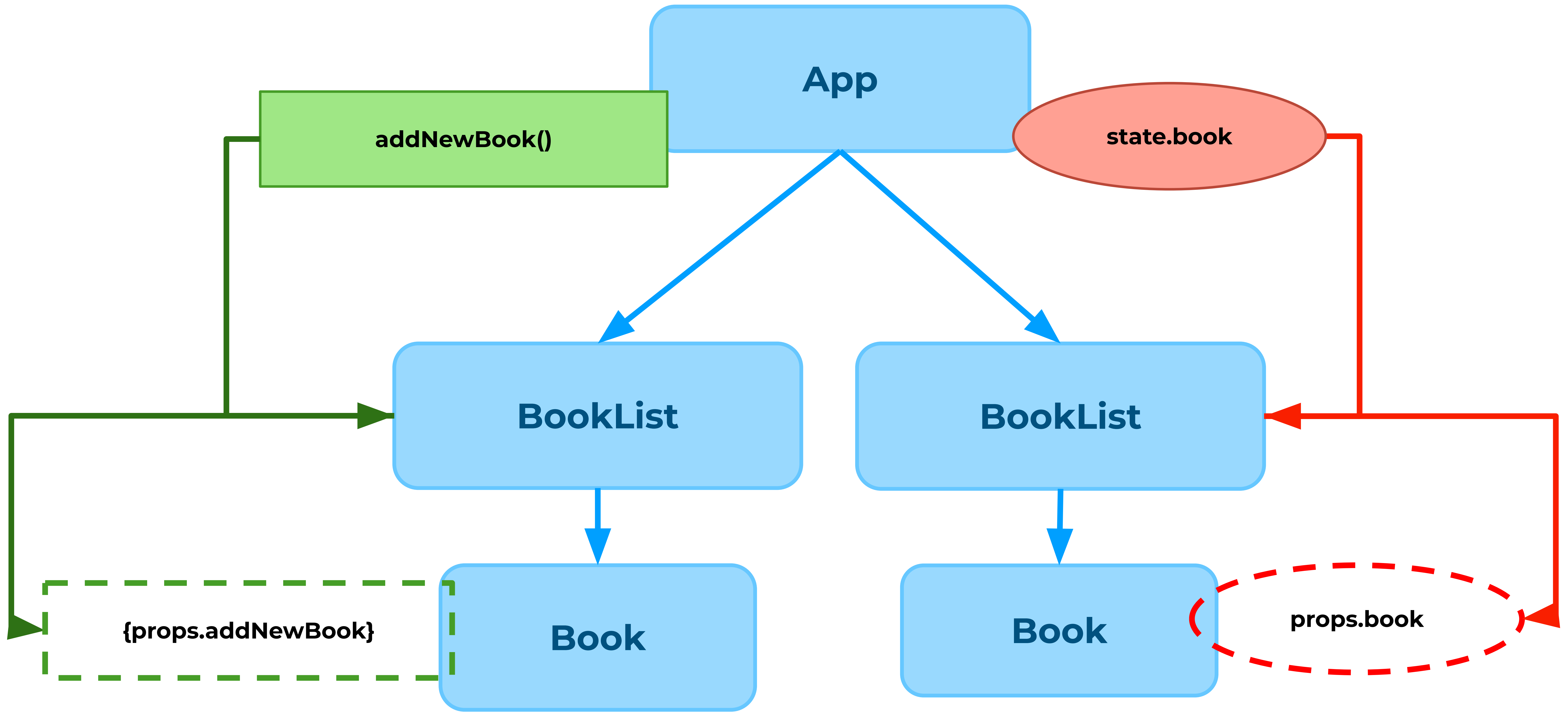
Lifting State Up

- To share a **state** between components the most common practice is to **lift state up** to a child component's most common ancestor. Removing **state** from the local component and **lifting it up** to the parent.



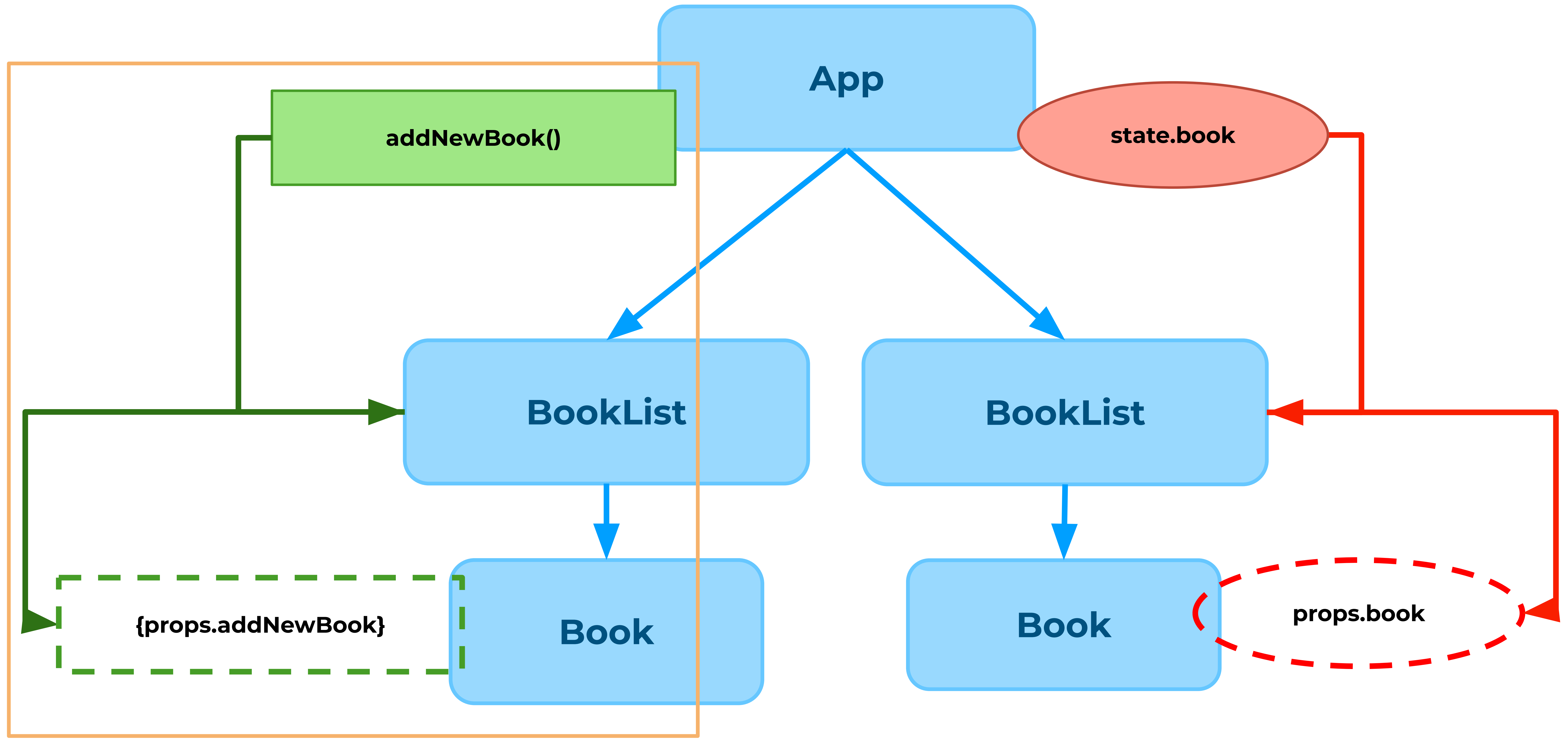
Lifting State Up

- This is commonly done to help with sharing **state** across multiple components. In the case of the **mybookcase** app, each BookList can have a list of books but as each book state changes the BookList will need to know about it.



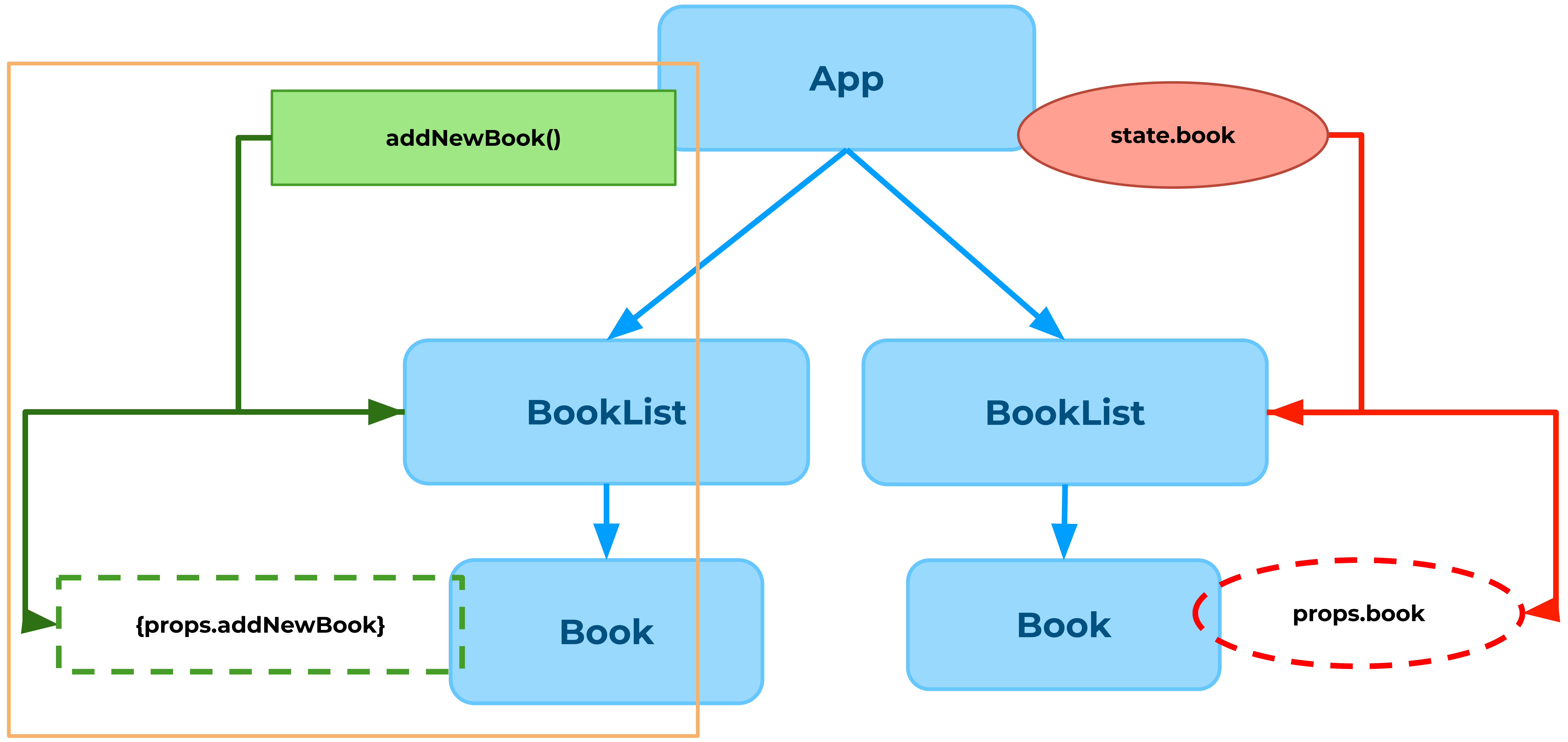
Lifting State Up

- As each Book has events associated with it you would need to lift the events up to the same parent/ancestor in order to change the relevant data when the event is triggered.



Lifting State Up

- Each event handle is then passed down to the relevant child via the **props**. In the **mybookcase** app the **books.json** file is referenced in the **App.js** and the state of the books are passed down through the props.



BLACK CODHER

CODING PROGRAMME

Exercise 4

Exercise 4: Events and useState

1. Open the **mybookcase** app

2. Open the Book.js file and copy the function addBook() from Book.js into App.js e.g.

```
const App = (props) => {
  const [books] = useState(data);

  function addBook(title) {
    console.log(`The Book ${title} was clicked`);
   }

  return  <BookList books={books} />
}
```

Exercise 4: Events and useState

3. Remove a book from the list when the "Add +" button is clicked. This will involve passing the id of the book to the addBook function in App.js

4. Calling the useState() hook and passing out a method (e.g. setBook) which you can call in your addBook method to update the books list

5. It is possible to pass event handlers down from the **App** component down to the **BookList** component through their properties

6. Set the attribute `addBook={addBook}` on the **BookList** component in the **App.js** file

Exercise 4: Events and useState

8. We can now pass the event handler from the BookList component down to the Book by setting the same attribute `addBook={props.addBook}` on the Book component in the **BookList.js** file

9. To finish passing down the event handler update the onClick handler on the button in Book.js with the handler code `onClick={() => props.addBook(id,title)}`

Exercise 4: Events and useState

Example addBook function in the App component (App.js file):

```
const App = (props) => {

  const [books,setBooks] = useState(data);

  function addBook(id,title) {
    console.log(`The Book ${title} was clicked.`);
    setBooks([...books.map(book => {
      // Add some conditional logic to either remove a book or
      // set a variable e.g. read=true
      // If setting a variable you will need to add some
      // conditional logic to your Book or BookList component to not display the book
    }
    )]);
}

  return  <BookList books={books} addBook={addBook} stored="library" />
}
```

Checkpoint!

How are you feeling?

RED - I have no idea what you're talking about.

YELLOW - I have some questions but feel like I understand some things.

GREEN - I feel comfortable with everything you've said.

Session 5 Summary

1. Reviewing Bookcase App Homework
2. Nested components
3. Using Proptypes for better validation and type checking
4. Lifting State Up
5. Handling Events
6. Continuing with the Bookcase App